Efficient Task Scheduling Algorithm for Multi-Core Systems

Nithin Gowda K N[1], [1]BMS College of Engineering, Bengaluru, nithinkn.lvs19@bmsce.ac.in
K P Lakshmi[2], [2]BMS College of Engineering, Bengaluru, kpl.ece@bmsce.ac.in

Abstract— A multi-core processor is a single integrated socket containing more than one processing unit, commonly known as cores. Multicore technology has become very popular and needed in recent years because of its wide range of advantages over single-core processors. Multicore processing has advantages in Energy Efficiency, Performance, Isolation, Reliability and Robustness and lesser hardware Costs. Automotive electronics are any electrically generated systems used in all sorts of automotive systems to control engines. Electronic Control Unit (ECU) is an embedded electronic device that reads signals coming from various sensors placed at various parts of the devices and controls important units like any automated operations. The overheads faced in multicores include contention of shared resources, synchronization between cores, and the overhead of scheduling jobs to those cores present. The proposed Life-Time(LT) scheduler framework is an approach of scheduling multicores in an efficient manner by addressing various multicore overheads and mainly focusing on the distribution of workload among the cores for their balanced performance at the same rate, keeping the other parameters almost comparable with the standard schedulers such as EDF, RM, LLF and G_FL scheduling algorithms.

Keywords: EDF - Earliest Deadline First, RM - Rate Monotonic, LLF - Least Laxity First, G_FL - Global-Fair Lateness.

1 Introduction

A single-core processor has to run many instructions in a limited amount of time, which demands to increase its speed of execution and that is out of normal mode, which in turn increases the heat dissipation and hence the performance, or integrity of the processor will be compromised. Efficient algorithms should be used for the implementation of cores to achieve better performance. Software that can run parallel is preferred. Multicore systems are used widely in various domains including Automotive Electronics.

Concentrating on vehicle automation, automotive electronics are any electrically generated systems used in road vehicles to control engines. The first electronic pieces used to control engine functions are referred to as Engine Control Units (ECU). A modern car may have up to 100 ECUs and a commercial vehicle up to 40. Electronic Control Unit (ECU) is an embedded electronic device that reads signals coming from various sensors placed at various parts of the car and controls important functions like any automated operations. A software supervisor is a computer system that controls different applications running simultaneously in the vehicle. Real-time requirements are mandatory in the automotive domain and are to be respected.

In this paper, cluster-based scheduling of tasks is proposed where the total number of tasks entered by the user is first grouped and then scheduled based on their cluster. The SimSo tool is used for scheduling purposes with scripts written in python. Random test cases are generated to schedule on the custom Life Time (LT) Scheduler and the test result is compared with the performance of the standard schedulers such as Earliest Deadline First (EDF), Rate Monotonic (RM), Least Laxity First (LLF) and Global-Fair Lateness (G-FL).

The overheads faced in multicores include contention of shared resources, synchronization between cores, and the overhead of scheduling jobs to those cores present. It may seem that by increasing the number of cores we can achieve parallelism and hence by reducing the load and execution becomes faster but as the number of cores is increased without proper control over it then overhead multicores become more dominant and affect the performance of the multicore system negatively. So, it is not correct to think that a multicore system can be made faster just by adding more cores. The number of cores that can be added should be able to balance so that it decreases the overheads on a single core system and it doesn't possess significantly higher overheads of the multicores. Thus, proper research or study on scheduling algorithms is needed to solve the overheads faced by the multicores.

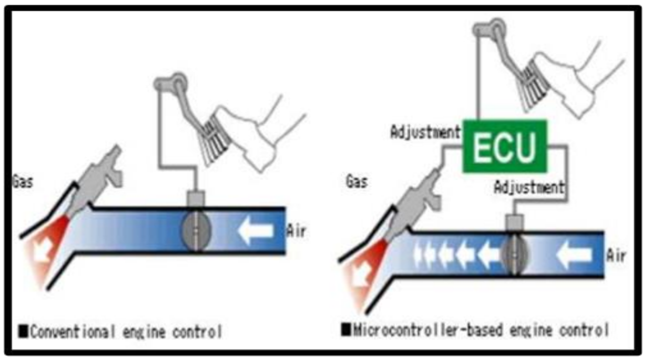

Gas
Air
Conventional engine control
Adjustment
ECU
Adjustment
Gas
Air
Microcontroller-based engine control

Fig 1. ECU Braking System

Non-linear growth in acceleration and deceleration when throttle or brake is applied so that greater change in speed can be seen with small human intervention as the intensity of throttle or brake is increased over time which is not easy to achieve without an Electronic Control Unit (ECU) in the braking system.

2 Standard Algorithms

In a real-time system, a scheduling algorithm decides the sequence in which tasks need to be executed.

2.1 Earliest Deadline First

In Earliest Deadline First (EDF) scheduling, at every scheduling point the task having the shortest deadline is taken up for scheduling. The Earliest deadline first selects a task according to its deadline such that a task with the earliest deadline has higher priority than others. It means the priority of a task is inversely proportional to its absolute deadline. EDF is an optimal algorithm which means if a task set is feasible, it is surely scheduled by EDF. Another thing is that EDF does not specifically take any assumption on periodicity of tasks so it is independent of the Period of task and therefore can be used to schedule aperiodic tasks as well.

2.2 Rate Monotonic

Rate- Monotonic Algorithm uses a mathematical model of static priority-based scheduling where the priority is the period of the tasks. It supports the intuition that the tasks that occur more frequently should be given higher priority. The Working of RMA depends on the periods of the tasks. These feasibility tests are generally known as schedulability bounds. For an algorithm to work, it must be within certain limits. Schedulability bound provides this limit.

IJSER

2.3 Least Laxity First

The Least Laxity First(LLF) scheduling algorithm assigns a priority to a task according to its executing urgency. The smaller the laxity value of a task is, the higher priority it will get and the sooner it gets executed. LLF algorithm also results in more task preemption as the laxity of each task changes at every instant of time and if a task other than the presently executing task gets lesser laxity then the present task will be preempted to schedule the other task which got lesser laxity.

2.4 G_FL

G-FL is a G-EDF-like scheduler but has lower maximum lateness bounds than GEDF. Due to its G-EDF like nature, it can be used within existing systems that implement arbitrary-deadline [14].

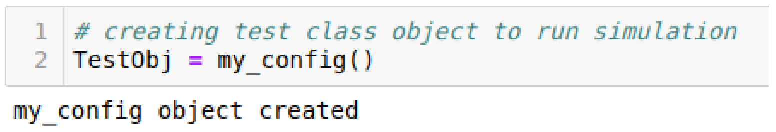
3 Proposed Technique

In this section, the execution flow of the task sets using the Life-Time scheduler logic is defined.

Step 1: Test Object Creation

All the necessary building blocks for the execution are gathered in one python class called **“my_config”**. So, the first thing a user needs to do is create an object for this class and then explore the inner possibilities to add and execute a test. It is very convenient to have all the building blocks available in one class.

Example:



```
# creating test class object to run simulation
TestObj = my_config()
```

```
my_config object created
```

Fig 2. Test object ‘TestObj’ creation

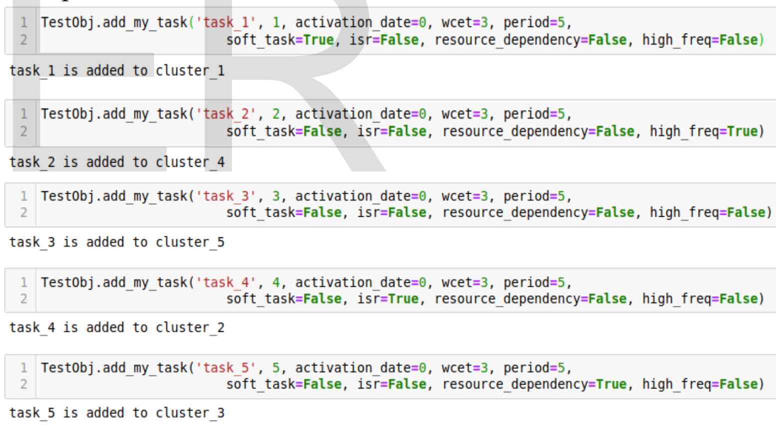
Step 2: Tasks Addition & Distribution

The tasks that are added are classified into any one of the five different clusters available based on the task parameters given at the time of adding tasks into the framework.

They are namely,

1. Application Software tasks
2. ISR (Interrupt Service Routine) tasks
3. Resource dependent tasks
4. High-frequency tasks
5. Low-frequency tasks

Example:



```
TestObj.add_my_task('task_1', 1, activation_date=0, wcet=3, period=5,
                    soft_task=True, isr=False, resource_dependency=False, high_freq=False)
```

```
task_1 is added to cluster_1
```

```
TestObj.add_my_task('task_2', 2, activation_date=0, wcet=3, period=5,
                    soft_task=False, isr=False, resource_dependency=False, high_freq=True)
```

```
task_2 is added to cluster_4
```

```
TestObj.add_my_task('task_3', 3, activation_date=0, wcet=3, period=5,
                    soft_task=False, isr=False, resource_dependency=False, high_freq=False)
```

```
task_3 is added to cluster_5
```

```
TestObj.add_my_task('task_4', 4, activation_date=0, wcet=3, period=5,
                    soft_task=False, isr=True, resource_dependency=False, high_freq=False)
```

```
task_4 is added to cluster_2
```

```
TestObj.add_my_task('task_5', 5, activation_date=0, wcet=3, period=5,
                    soft_task=False, isr=False, resource_dependency=True, high_freq=False)
```

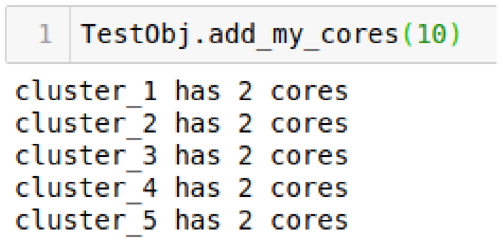
```
task_5 is added to cluster_3
```

Fig 3. Adding tasks to the TestObj object

Step 3: Cores Addition & Distribution

Once the overall tasks are added they are classified into clusters, the total number of cores for execution is specified and they are distributed among the available clusters proportionally based on the total Worst-Case Execution Time (WCET) within which all tasks in each cluster will be executed.

Example:



```
TestObj.add_my_cores(10)
```

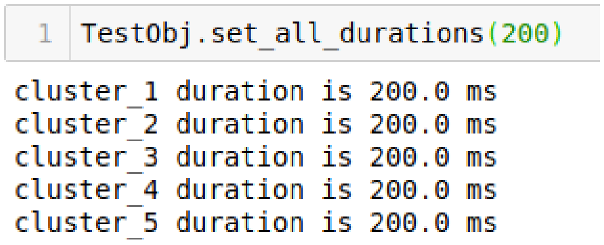
```
cluster_1 has 2 cores
cluster_2 has 2 cores
cluster_3 has 2 cores
cluster_4 has 2 cores
cluster_5 has 2 cores
```

Fig 4. Adding cores to the TestObj object

Step 4: Specify Execution Time

Once both tasks and cores are added and distributed among the available clusters, the total execution time specified to run the simulation is specified to run the framework.

Example:



```
TestObj.set_all_durations(200)
```

```
cluster_1 duration is 200.0 ms
cluster_2 duration is 200.0 ms
cluster_3 duration is 200.0 ms
cluster_4 duration is 200.0 ms
cluster_5 duration is 200.0 ms
```

Fig 5. Adding Simulation Time to TestObj object

Step 5: Life-Time (LT) Scheduler

In this paper, we will introduce a new method of assigning priorities depending on each task's lifetime, where Life-time = Deadline-time - Arrival-time → (if the task is not yet running)
Lifetime = Deadline-time - present-time → (if the task is running)
If the lifetime is less than the priority will be high and if the lifetime is more than the priority will be less for a task. By considering the priorities, tasks will be preempted whenever necessary.

IJSER

Step 6: Run the simulation for each cluster of tasks

Once the setup is all done, we can run the simulation which internally runs as specified in the cluster manner with better cores distributions and task executions.

4 Data Analysis

The Simulations of this framework were tested using a python programming language by considering a variable number of task sets considered at each iteration in the order of 10, 20, 30 and 40 numbers of task sets running for a total of 200ms and with 10 cores. The framework is tested for the custom scheduler technique "Life-Time" scheduler, after which the results are compared with the standard scheduling algorithms namely Earliest Deadline First (EDF), Rate Monotonic (RM), Least Laxity First (LLF) and Global-Fair Lateness.

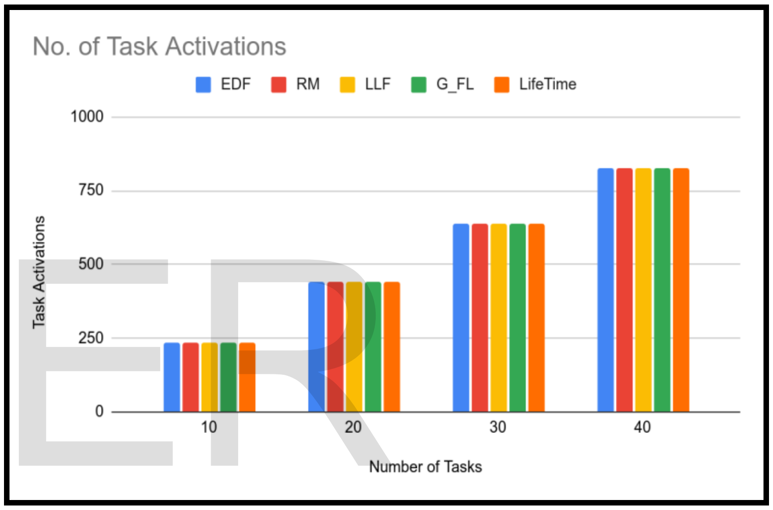
4.1 Tasks Activations Analysis

Task activation is defined as the instantiation of any instance of a task (periodic or aperiodic) that occurred in the execution and hence every activation of the instance of the periodic task is considered as one task activation.

A number of tasks activations are observed at each scheduler technique for the various task sets generated. This gives an overview of how many task instances were inserted, where each occurrence of a periodic task is considered as one instance of the task.

No. of Tasks Activations					
No. of Tasks	EDF	RM	LLF	G_FL	Life-Time
10	235	235	235	235	235
20	442	442	442	442	442
30	637	637	637	637	637
40	825	825	825	825	825

Table 1. No. of Tasks Activations



Graph 1. No. of Tasks Activations

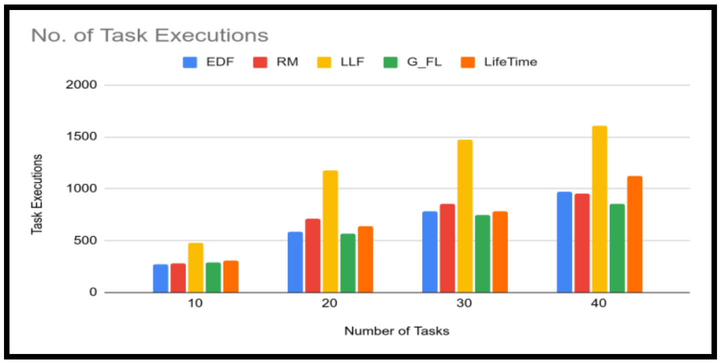
From the above analysis, it is evident that the same task sets were introduced to all the schedulers considered and hence the same number of task activations are observed for all schedulers in the particular scenarios.

4.2 Tasks Executions Analysis

A number of tasks executions are observed at each scheduler technique for the various task sets generated. This gives an overview of how many task scenarios were actually running in the simulation time as a whole. Each occurrence of a periodic task is considered as one instance of the task.

No. of Tasks Executions					
No. of Tasks	EDF	RM	LLF	G_FL	Life-Time
10	273	284	477	287	311
20	588	713	1175	573	644
30	781	858	1474	745	785
40	975	952	1608	851	1120

Table 2. No. of Tasks Executions



Graph 2. No. of Tasks Executions

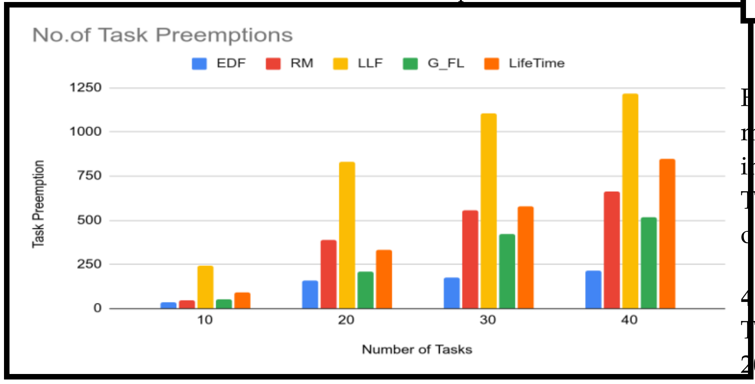
From the above analysis, it is evident that the Life-Time scheduler results are comparable to the rest of the scheduler techniques (EDF, RM, and G_FL) and better than LLF.

4.3 Tasks Preemptions Analysis

Task preemption is defined as the removal of a running instance of the task when the scheduler finds another better fit and more prior task instance that needs to be executed first than the present running task instance.
A number of task preemptions are observed at each scheduler technique for the various task sets generated.

No. of Tasks Preemptions					
No. of Tasks	EDF	RM	LLF	G_FL	Life-Time
10	38	49	244	52	94
20	159	391	831	209	335
30	175	557	1107	425	582
40	217	666	1220	520	850

Table 3. No. of Tasks Preemptions



Graph 3. No. of Tasks Preemptions

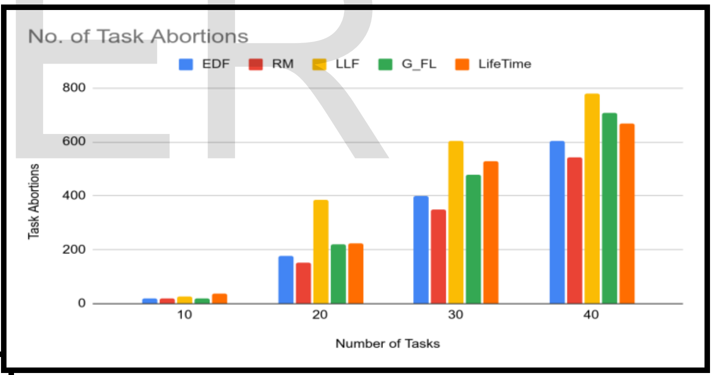
From the above analysis, it is evident that the Life-Time algorithm has more tasks preemptions when compared with EDF, RM and G_FL algorithms techniques and fewer task preemptions than the LLF algorithm technique. This result can also be related to the tasks executions analysis as proportional.

4.4 Tasks Abortions Analysis

Task Abortion is defined as when a task fails to get executed within its Worst-Case Execution Time (WCET) and so it will be removed from the scheduling process and marked as "Task Aborted" signifying that instance of the task is removed and did not complete execution.
A number of tasks abortions are observed at each scheduler technique for the various task sets generated which actually gives us the idea of how many tasks have failed to complete their work before the deadline. For a real-time scheduler, this number should be as minimum as possible.

No. of Tasks Abortions					
No. of Tasks	EDF	RM	LLF	G_FL	Life-Time
10	21	21	28	21	38
20	178	151	385	221	223
30	398	351	603	477	529
40	603	544	780	709	670

Table 4. No. of Tasks Abortions



Graph 4. No. of Tasks Abortions

From the above analysis, it is evident that the LLF algorithm has more number of tasks abortions observed for the same task sets in comparison with the other scheduler techniques. The Life-Time scheduler results in this analysis are compared with the other schedulers except and better than the LLF algorithm.

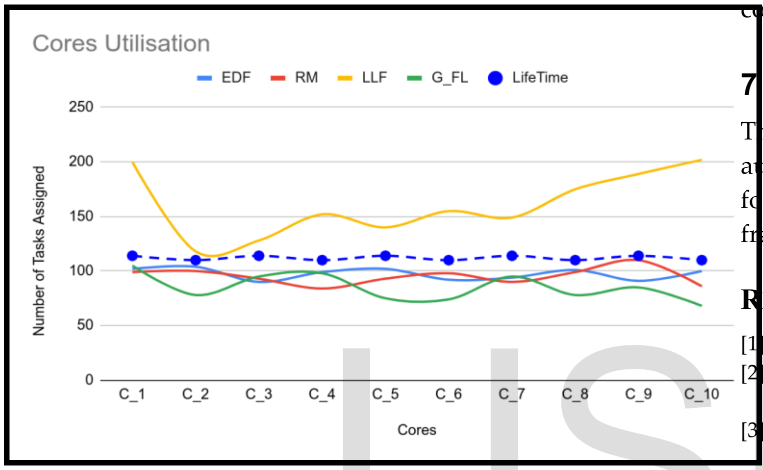
4.5 Cores Utilization Analysis

The scenario of task sets consisting of 40 tasks, 10 cores and 200ms of execution time are considered for the below analysis.

Cores Utilisation					
Cores	EDF	RM	LLF	G_FL	Life-Time
C_1	102	99	200	105	114
C_2	104	100	118	78	110

C_3	90	93	128	95	114
C_4	99	84	152	98	110
C_5	102	93	140	75	114
C_6	92	98	155	74	110
C_7	94	90	149	95	114
C_8	101	99	175	78	110
C_9	91	110	189	85	114
C_10	100	86	202	68	110

Table 5. Cores Utilization



Graph 5. Cores Utilization

From the above analysis, it is evident that the Life-Time scheduler has better core utilization compared to the other four (EDF, RM, LLF and G_FL) schedulers compared here, hence Life-Time scheduler can be a better performer in distributing the workload among the multicores that are available.

5 Results and Discussions

From the above data analysis, it is evident that the LLF algorithm has more number of tasks pre-emptions, tasks abortions and tasks executions are observed for the same task sets in comparison with the other scheduler techniques. The Life-Time (LT) scheduler results in a comparable set of analyses with the other schedulers. From the tasks activations analysis, it is evident that the same task sets were introduced to all the schedulers considered and hence the same number of task activations are observed for all schedulers in a particular scenario. From the aspect of cores utilization analysis, it is evident that Life-Time scheduler has better balanced and distributed core utilization compared to the other four schedulers (EDF, RM, LLF and G_FL) compared here, hence Life-Time scheduler can be a better performer in distributing the workload among the multicores that are considered in this framework.

6 Conclusion

The proposed framework gives the result in comparison with the other schedulers on the normal factors analysis of considering tasks pre-emptions, tasks abortions, tasks activations and tasks executions and hence by proving to be a framework with a customized scheduler which does not deviate much from the traditional approaches and findings. The proposed framework becomes promising and better than other compared scheduler frameworks when it comes to cores utilization in multicore setup. It shows that the proposed Life-Time scheduler framework can provide a better performer in distributing the workload among the multicores that are considered and available in the framework.

7 Limitations and Future Scope

The proposed custom scheduler framework is not completely automated to handle the random task generations and testing for quick usages and analysis. So, an automated setup of this framework can come in very handy in future if developed.

References

[1] M. Franklin, “Notes from ENEE759M: Microarchitecture”, Spring 2008
[2] D. Geer, “For Programmers, Multicore Chips Mean Multiple Challenges”, Computer, September 2007
[3] M. Creeger, “Multicore CPUs for the Masses”, QUEUE, September 2005
[4] R. Alderman, “Multicore Disparities”, VME Now, December 2007, http://vmenow.com/c/index.php?option=com_content&task=view&id=105&Itemid=46
[5] Geetishree Mishra, Rajeshwari Hegde, "Performance optimization of task intensive real-time applications on multicore ECUs - a hybrid scheduler", International Journal of Reconfigurable and Embedded Systems (IJRES), July 2019.
[6] G. Xie et al., "WCRT Analysis and Evaluation for Sporadic Message-Processing Tasks in Multicore Automotive Gateways," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 2, pp. 281-294, Feb. 2019, DOI: 10.1109/TCAD.2018.2812119.
[7] Y. Wang, X. Jiang, N. Guan, Z. Guo, X. Liu and W. Yi, "Partitioning-Based Scheduling of OpenMP Task Systems with Tied Tasks," in IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 6, pp. 1322-1339, 1 June 2021, DOI: 10.1109/TPDS.2020.3048373.
[8] J. Sun, N. Guan, Y. Wang, Q. He, and W. Yi, “Real-time scheduling and analysis of OpenMP task systems with tied tasks,” in Proc. IEEE Real-Time Syst. Symp., 2017, pp. 92–103
[9] Y. Li, L. Zhong and F. Lin, "Predicting-Scheduling-Tracking: Charging Nodes with Non-Deterministic Mobility," in IEEE Access, vol. 9, pp. 2213-2228, 2021, DOI: 10.1109/ACCESS.2020.3046857.
Chen Da-Ren, Chen Young-Long, Chen You-Shyang, Time and Energy Efficient DVS Scheduling for Real-Time Pinwheel Tasks, Journal of Applied Research and Technology, Volume 12, Issue 6, 2014, Pages 1025-1039, ISSN 1665-6423, https://doi.org/10.1016/S1665-6423(14)71663-3.
[10] O. Abid, Q. Cabannes and B. Senouci, "Supervisor and control investigation in smart/autonomous vehicles: Environment recognition and objects detection ADAS application case study," 2018 11th

IJSER

International Symposium on Mechatronics and its Applications (ISMA), 2018, pp. 1-7, DOI: 10.1109/ISMA.2018.8330135.

[11] Embedded Systems in Automobiles- A Boon to Automobile Industry https://www.mepits.com/tutorial/299/Embedded-System/EmbeddedSystemsin-Automobiles—A-Boon-to-Automobile-Industry, Published on 20 February 2015.

[12] https://cecas.clemson.edu/cvel/auto/systems/airbag_deployment.html

[13] Chéramy, Maxime & Hladik, Pierre-Emmanuel & Déplanche, Anne-Marie. (2014). SimSo: A Simulation Tool to Evaluate Real-Time Multiprocessor Scheduling Algorithms.

[14] https://www.cs.unc.edu/~anderson/papers/ecrts12c.pdf

IJSER